Midterm Review

- Practice material from the Winter 2014 midterm.
- Will cover some (but not all) of the questions.
- Midterm content: Everything up to / including modules.
Short Answer

List three advantages of modularization, and briefly explain.
Short Answer

List three advantages of modularization, and briefly explain:

- **Re-usability**: Build programs faster, and build large programs more easily.

- **Maintainability**: Makes debugging and changing a program easier.

- **Abstraction**: No need to know / understand all parts to use them. Allows changing internals without breaking things.
Clicker Question

For positive integers, the C modulo operator (%) behaves the same as the Racket quotient function.

A  True.
B  False.
C  Which one is quotient again?
Clicker Question

For positive integers, the C modulo operator (%) behaves the same as the Racket quotient function.

A  True.
B  *False.
C  Which one is quotient again?

The modulo operator behaves the same as the remainder function.
Short Answer

Briefly explain the special form \textit{provide} in Racket, and why there is no equivalent in C.
Short Answer

Briefly explain the special form `provide` in Racket, and why there is no equivalent in C:

- Gives identifiers “program scope” or specifies bindings available in a module.

- All functions and variables in C have program scope by default (and are therefore available to other modules).
Clicker Question

Every C module must have a main function.

A True.
B False.
C True, but only on Tuesdays.
Clicker Question

Every C module must have a main function.

A  True.
B  *False.
C  True, but only on Tuesdays.

A module does not require a main function. (Example: The stack module from the last assignment).
Short Answer

Racket uses *dynamic typing*. What kind of typing does C use? Give one advantage of the typing C uses. Provide a brief Racket example that demonstrates dynamic typing that isn’t possible in C.
Short Answer

Racket uses *dynamic typing*. What kind of typing does C use? Give one advantage of the typing C uses. Provide a brief Racket example that demonstrates dynamic typing that isn’t possible in C:

- C uses static typing.
- Advantage: Built-in contract, can detect type errors at compile-time, fast (no run-time type checking).
- Example (there are many):
  
  (define dyntype (if (>= x 0) x “invalid”))
Clicker Question

In C, \((a \neq 0) \land (b/a == 2)\) will produce an error if \(a\) is 0.

A  True.
B  False.
C  Impossible to know without trying it.
In C, \((a \neq 0) \&\& (b/a == 2)\) will produce an error if \(a\) is 0.

A  True.
B  *False.
C  Impossible to know without trying it.

Because \&\& is used, the first condition will short circuit the evaluation, preventing \(b/a\) from being executed.
Short Answer

Briefly explain the purpose of *requires* and *effects* function documentation in C.
Short Answer

Briefly explain the purpose of *requires* and *effects* function documentation in C:

- Requires: Identifies conditions that must be true when calling the function (e.g. restrictions on parameters).
- Effect: Identifies what a function prints, reads, or mutates.
Clicker Question

`printf("hello!\n")` is a C expression with a value of 7.

A True.

B False.

C The only “expressions” in C are my cries of pain when I’m forced to code in it.
Clicker Question

printf("hello!\n") is a C expression with a value of 7.

A  *True.
B  False.
C  The only “expressions” in C are my cries of pain when I’m forced to code in it.

The return value of printf is the number of characters printed.
Short Answer

Write the declaration for a C function `add` that takes two ints, `x` and `y`, and returns an int `x+y`. 
Write the declaration for a C function add that takes two ints, x and y, and returns an int x+y:

- No actual addition required!

- \( \text{int add(int x, int y);} \)
Clicker Question

In the following C code, the assignment operator appears only once:

```c
bool nisfive = (n == 5);
```

A True.
B False.
C This has to be harder than it looks but I’m not seeing the trick.
Clicker Question

In the following C code, the assignment operator appears only once:

```
bool nisfive = (n == 5);
```

A  True.
B  *False.
C  *This has to be harder than it looks but I’m not seeing the trick.

The code above uses = for initialization, NOT assignment. They are very slightly different (e.g. some struct-related notation can only be used during initialization, and not assignment).
Coding Question

Write the C function `pyramid(int n);` that prints a pyramid of numbers with `n` lines. You may assume `n` is an odd integer in the range [3-99]. Use the printf formatter “%3d” to print out each integer. `pyramid(7)` prints the following:

```
1
2   3
3   4   5
4   5   6   7
3   4   5
2   3
1
```
Coding Question

General approach:

- Need nested for loops / while loops.
- One set of loops to count from row 1 to \( n/2 + 1 \), one set to count back down from row \( n/2 \) to 1.
void pyramid(int n) {
    for (int i=1; i<=n/2+1; ++i) {
        for (int j=i; j<=2*i-1; ++j) {
            printf("%3d",j);
        }
        printf("\n");
    }

    for (int i=n/2; i>0; --i) {
        for (int j=i; j<=2*i-1; ++j) {
            printf("%3d",j);
        }
        printf("\n");
    }
}

Stack Trace

Draw the call stack immediately after “In exchange: ” is printed. For pointers, draw an arrow to the variable they point at.
void exchange(int *pa, int* pb) {
    int temp;
    temp = *pa;
    *pa = *pb;
    *pb = temp;
    printf("In exchange: ");
    \[⇒\] printf("a = %d, b = %d\n", *pa, *pb);
}

int main(void) {
    int a, b;
    a = 5;
    b = 7;
    printf("In main: ");
    printf("a = %d, b = %d\n", a, b);
    exchange(&a, &b);
    printf("a = %d, b = %d\n", a, b);
}
Stack Trace

exchange:
   pa: [arrow pointing to a]
   pb: [arrow pointing to b]
   temp: 5
   return addr: main : 16

==========

main:
   a: 7
   b: 5
   return addr: OS